

First Hit    Fwd Refs

Generate Collection

Print

L12: Entry 1 of 2

File: USPT

Jan 21, 2003

DOCUMENT-IDENTIFIER: US 6510437 B1

TITLE: Method and apparatus for concurrent thread synchronization

Brief Summary Text (10):

In a concurrent threading model, multiple threads are allowed to execute independently of one another. Rather than being cooperatively scheduled like cooperative threads, concurrent threads are preemptively scheduled. That is, a computation by a given concurrent thread may be preempted at any point in time by an outside entity such as another concurrent thread, e.g., a scheduler, or the operating system. Thread preemption may occur because of meaningful events in the execution of a program. By way of example, a meaningful event may be when a thread's priority is programmatically raised to be higher than that of a currently running thread. Alternatively, thread preemption may occur because of artificially induced events such as the elapsing of a particular interval of time.

Brief Summary Text (11):

During the execution of an object-based program, a thread may attempt to execute operations which involve multiple objects. On the other hand, multiple threads may attempt to execute operations which involve a single object. Frequently, only one thread is allowed to invoke one of some number of operations, i.e., synchronized operations, that involve a particular object at any given time. That is, only one thread may be allowed to execute a synchronized operation on a particular object at one time. A synchronized operation, e.g., a synchronized method, is block-structured in that it requires that the thread invoking the method to first synchronize with the object that the method is invoked on, and desynchronize with that object when the method returns. Synchronizing a thread with an object generally entails controlling access to the object using a synchronization construct before invoking the method.

Brief Summary Text (13):

Since a concurrent thread is not able to predict when it will be forced to relinquish control, synchronization constructs such as locks, mutexes, semaphores, and monitors may be used to control access to shared resources during periods in which allowing a thread to operate on shared resources would be inappropriate. By way of example, in order to prevent more than one thread from operating on an object at any particular time, objects are often provided with locks. The locks are arranged such that only the thread that has possession of the lock for an object is permitted to execute a method on that object. With respect to FIG. 1, a process of acquiring an object lock will be described. The process of acquiring an object lock begins at step 104 where a thread obtains the object on which the thread wishes to operate. In general, the object on which the thread intends to operate has an associated object lock. Then, in step 106, a determination is made regarding whether the object is locked. That is, a determination is made regarding whether the object lock associated with the object is held by another thread, e.g., a thread that is currently operating on the object.

Drawing Description Text (5):

FIG. 3a is a diagrammatic representation of a cooperative thread stack and an object in accordance with an embodiment of the present invention.

Drawing Description Text (6):

FIG. 3b is a diagrammatic representation of the cooperative thread stack and the object of FIG. 3a after the header value of the object has been placed on the stack in accordance with an embodiment of the present invention.

Drawing Description Text (7):

FIG. 3c is a diagrammatic representation of the cooperative thread stack and the object of FIG. 3b after the object has been re-entered by the thread associated with the thread stack in accordance with an embodiment of the present invention.

Drawing Description Text (8):

FIG. 4a is a process flow diagram which illustrates the steps associated with acquiring an object lock using a cooperative thread in accordance with an embodiment of the present invention.

Drawing Description Text (9):

FIG. 4b is a process flow diagram which illustrates the steps associated with unlocking an object locked using a cooperative thread in accordance with an embodiment of the present invention.

Detailed Description Text (10):

As previously mentioned, in one embodiment, threads may either be cooperative or concurrent. Cooperative threads differ from concurrent threads in that once a cooperative thread has control, the cooperative thread maintains control until the cooperative thread voluntarily relinquishes control. On the other hand, when a concurrent thread has control, the concurrent thread may lose control at any time.

Detailed Description Text (11):

In general, in order for a thread to execute a synchronized operation on an object, the thread obtains the lock associated with the object. In one embodiment, obtaining an object lock involves obtaining the value of the object header field. When a cooperative thread obtains an object lock, the cooperative thread holds the object lock until the cooperative thread has completed its use of the object, without interference from other threads. The steps associated with a cooperative thread obtaining an object lock will be described below with reference to FIG. 4a, whereas the steps associated with a concurrent thread obtaining an object lock will be described below with respect to FIGS. 6a and 6b.

Detailed Description Text (12):

With reference to FIG. 4a, a process of acquiring an object lock by a cooperative thread will be described in accordance with an embodiment of the present invention. In the described embodiment, the process of acquiring an object lock involves obtaining the value of the object header field, as mentioned above. The process begins at step 402 in which the thread reads the contents of the header field of the object. Then, in step 404, using the object header field contents, a determination is made regarding whether the object is locked. In general, when an object is locked, the object header field contents include a forwarding pointer, or a reference, to a thread which has locked the object. Alternatively, when an object is unlocked, the object header field contents include the header value of the object.

Detailed Description Text (15):

When a synchronized operation returns, the thread, e.g., the cooperative thread, which holds an object lock no longer needs the object associated with the object lock. Hence, the cooperative thread unlocks the object. Upon return or deactivation of the synchronized operation, when the header value of the object is encountered on the stack, the header value is stored over the forwarding pointer in the object header field. By returning the header value to the object header field, the object is unlocked, i.e., made free to be locked. Alternatively, when an indicator value reflecting a re-entrant lock acquisition is encountered on the thread stack when an

attempt is made to unlock a locked object, the stack is "popped," as will be appreciated by those skilled in the art. Because the header value of the object remains on the thread stack in this latter case, the thread continues to hold the lock on the object until an unlocking operation is performed that returns the header value to the object header field.

Detailed Description Text (16):

In a cooperative threading model, a thread is allowed to maintain control until the thread voluntarily gives control to some other cooperative thread. Thus, once a cooperative thread has begun the operation of acquiring a lock on an object, this first cooperative thread may ensure that no second cooperative thread will gain control and be permitted to run until the first cooperative thread has completed acquiring the lock. Alternatively, if the lock on the object is already owned by some other cooperative thread, the first cooperative thread may ensure that no second cooperative thread will gain control and be permitted to run until the first cooperative thread has arranged to wait for the lock on the object to become free.

Detailed Description Text (18):

FIG. 4b is a process flow diagram which illustrates the steps associated with a cooperative thread unlocking an object which it has previously locked in accordance with an embodiment of the present invention. The process begins at step 422 where the object header field contents are read from the object which is to be unlocked. It should be appreciated that the same cooperative thread which is attempting to unlock the object has previously locked the object. Once the contents of the object header field are obtained, a determination is made in step 424 regarding whether the lock on the object is a re-entrant lock. In other words, a determination is made as to whether the contents of the object header field indicate that the object is also locked by a previous synchronized operation invoked by the thread on the object, in addition to the current synchronized operation which is associated with unlocking the object.

Detailed Description Text (19):

If it is determined that the lock is re-entrant, then in step 426, the indicator value which indicates that the lock is being used by a re-entrant synchronized operation is popped from the stack. As previously mentioned, the indicator value may be any suitable value which indicates that the header value of the object is stored in another location on the thread stack. Once the indicator value is removed, with respect to the current synchronized operation, the process of unlocking the object is considered to be completed. It should be appreciated, however, that the object is still locked by the same cooperative thread with respect to a previous synchronized operation performed by the same cooperative thread on the object.

Detailed Description Text (46):

In a concurrent threading model, when several concurrent threads are concurrently trying to study an object, the priorities assigned to the concurrent threads often affects when and if a particular thread will be allowed to study the object. In general, threads with a higher execution priority will obtain the right to study the object before threads with a lower execution priority will obtain the right to study the object. However, it is possible that a thread with a lower priority will initially obtain the right to study an object, then be forced to give up control, i.e., are preempted, because a thread with a higher priority has become "runnable." In effect, the thread with the lower priority thus prevents the thread with a higher priority from studying the object, because the thread with the higher priority, upon attempting to swap a sentinel value for the contents of the object header field, will find the sentinel value of the thread with the lower priority in the object header field. At the same time, the thread with the lower priority is unable to execute and thus finish studying the object because the thread with the higher priority is runnable and so will be run in favor of the thread with the lower priority. As a result, neither the thread studying the object nor the thread

desiring to study the object is able to make progress.